CMPS160 : Notes on the Terrain Rendering Lab

Section 1 : First steps in texture mapping

These notes are not meant to be a comprehensive introduction to texture mapping. Rather they are meant to get you up and running quickly so you will have a working example. This lab does not have any special starter code, use the supplied code fragments together with your existing opengl code from lab3.

To work with OpenGL textures, you must first load your image and bind it to a unique texture Id. Then you must specify the appropriate texture coordinate for each vertex. Download the 'skin.ppm' (for texturing) and 'map.ppm' (for height maps) from the lab website and put them in your build folder.

1) Loading and binding the texture

We need to keep track of the texture ids that we use. In this example we have only one, so make a single element array in our class (GLWidget) declaration.

```cpp
GLuint texture[1]; // Storage For One Texture
```

Now we can load and bind the texture. The initializeGL() function is a good place to do this. We have an option here to do the simplest texture loading, or use MipMapping. I'll give examples of both. Feel free to try things the first way for getting a feel for things but I heavily recommend switching to MipMapping for the actual project.

1. Quick way to load and bind a texture in Qt GLWidget

```cpp
texture[0]=bindTexture(QImage("map.ppm")); // Use Qt convenience method
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

glEnable(GL_TEXTURE_2D); //Enable Texture Mapping (Needed to display textures)
```

2. Using MipMapping

```cpp
QImage i;
// Load Image
if (i.load("map.ppm"))
    qDebug("Image loaded successfully");

// Convert our QImage to an OpenGL friendly format
i = QGLWidget::convertToGLFormat( i );
// Generate a new unique texture id and put it in texture[0]
glGenTextures( 1, &texture[0] );
// Activate the texture
glBindTexture( GL_TEXTURE_2D, texture[0] );

// Texture filtering : Note special filter which uses MipMaps
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```
// build our texture mipmaps
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, i.width(), i.height(),
    GL_RGBA, GL_UNSIGNED_BYTE, i.bits());

glEnable(GL_TEXTURE_2D); // Enable Texture Mapping

2) Drawing texture mapped primitives

Once the texture is bound, we must tell OpenGL when to start drawing with that texture. After that all you need to do is specify the texture coordinate for each vertex. The following code does exactly that, and you can use it in paintGL()

glBindTexture(GL_TEXTURE_2D, texture[0]); // Select Our Texture

// draw four tiles and texture them, tiles will be on the Z plane(Y=0) and span (0,0,0)-(2,0,2)
gBegin(GL_QUADS);
    // first tile (0,0,0)-(1,0,1), mapped with the first quadrant of the texture (0,0)-(0.5,0.5)
        // the vertices are arranged in clockwise order
        glVertex3f(0.0f, 0.0f, 0.0f); glVertex3f(0.0f, 0.0f, 0.0f);
        glVertex3f(1.0f, 0.0f, 1.0f); glVertex3f(1.0f, 0.0f, 1.0f);

    // second tile (1,0,0)-(2,0,1), mapped with the second quadrant of the texture (0.5,0)-(1.0,0.5)
        glVertex3f(0.5f, 0.0f, 0.0f); glVertex3f(1.0f, 0.0f, 0.0f);
        glVertex3f(1.0f, 0.5f, 0.0f); glVertex3f(2.0f, 0.0f, 0.0f);

    // third tile (0,0,1)-(1,0,2), mapped with the third quadrant of the texture (0,0.5)-(0.5,1.0)
        glVertex3f(0.0f, 0.5f, 0.0f); glVertex3f(0.0f, 0.0f, 1.0f);
        glVertex3f(0.5f, 1.0f, 0.0f); glVertex3f(2.0f, 0.0f, 2.0f);

    // fourth tile (1,0,1)-(2,0,2), mapped with the fourth quadrant of the texture (0.5,0.5)-(1.0,1.0)
        glVertex3f(0.5f, 0.5f, 0.0f); glVertex3f(1.0f, 0.0f, 1.0f);
        glVertex3f(1.0f, 1.0f, 0.0f); glVertex3f(2.0f, 0.0f, 2.0f);

glEnd();

Note that the Y coordinates for the vertices was always zero. This example has a flat mesh. For the lab you'll have to change this to a height-field, based on the intensity values in the image 'map.ppm'